TABULAR PLAYGROUND SERIES – APR 2021

The goal of these competitions is to provide a fun, and approachable for anyone, tabular dataset. These competitions will be great for people looking for something in between the Titanic Getting Started competition and a Featured competition. The dataset is used for this competition is synthetic but based on a real dataset and generated using a CTGAN.

DATASET

- In this competition, we will gain access to two similar datasets that include passenger information like name, age, gender etc. One dataset is titled train.csv , the other is titled test.csv. and Sample_submission.csv
- Train.csv will contain the details of a subset of the passengers on board (891 to be exact) and importantly, will reveal whether they survived or not, also known as the "ground truth".
- The test.csv dataset contains similar information but does not disclose the "ground truth" for each passenger. It's your job to predict these outcomes.

The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)

The **training set** should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class.

The **test set** should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Synthanic.

Goal

The task is to predict whether or not a passenger survived the sinking of the Synthanic (a synthetic, much larger dataset based on the actual Titanic dataset). For each PasengerId row in the test set, you must predict a 0 or 1 value for the Survived target.
Your score is the percentage of passengers you correctly predict. This is known as accuracy.

Data Dictionary

Variable	**Definition**	**Key**
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	

sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes

pclass: A proxy for socio-economic status (SES)
1st = Upper
2nd = Middle
3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

sibsp: The dataset defines family relations in this way...

Sibling = brother, sister, stepbrother, stepsister

Spouse = husband, wife (mistresses and fiancés were ignored)

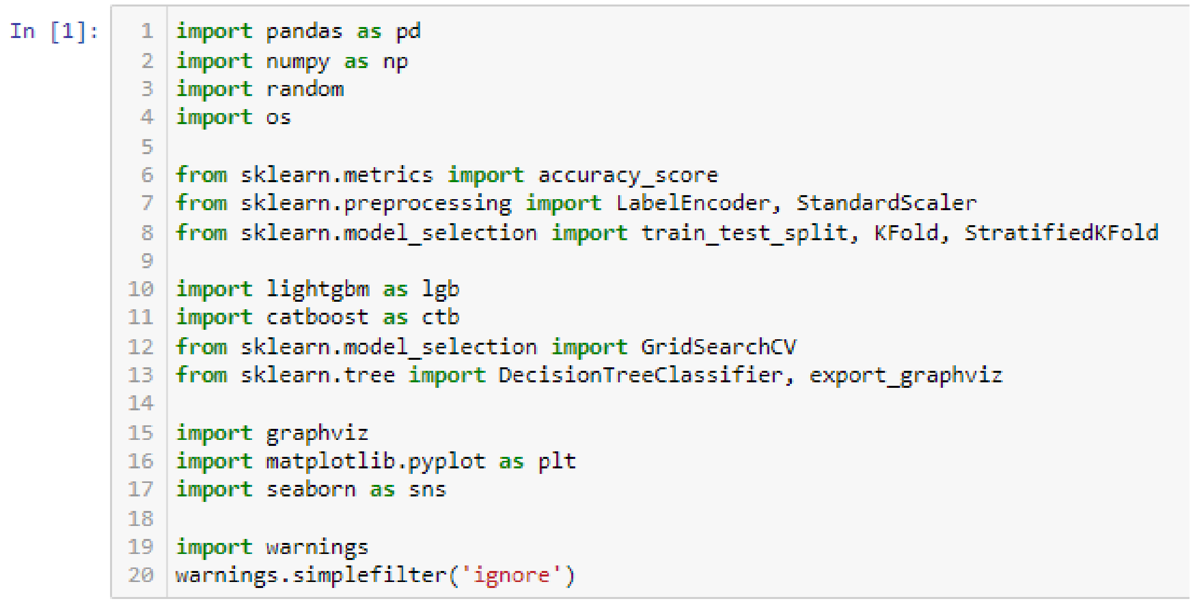
parch: The dataset defines family relations in this way...

Parent = mother, father

Child = daughter, son, stepdaughter, stepson

Some children travelled only with a nanny, therefore parch=0 for them.

Step 1: Importing the necessary libraries



```
In [1]:
import pandas as pd
import numpy as np
import random
import os

from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold

import lightgbm as lgb
import catboost as ctb
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier, export_graphviz

import graphviz
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter('ignore')
```

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

Accuracy score is multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_true.

K-Folds cross-validator : Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Stratified K-Folds cross validation : Provides train/test indices to split data in train test sets.

This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

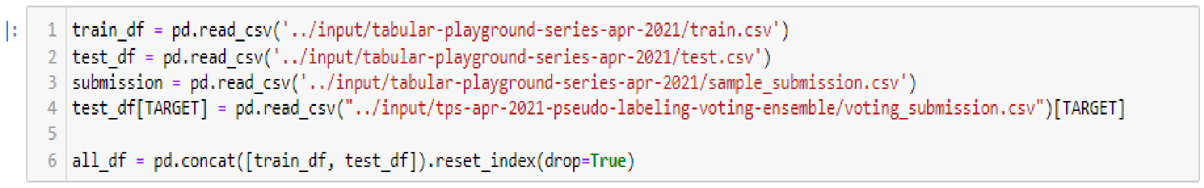
LightGBM is a fast, distributed, high performance gradient boosting framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks.

Important members are fit, predict.

GridSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

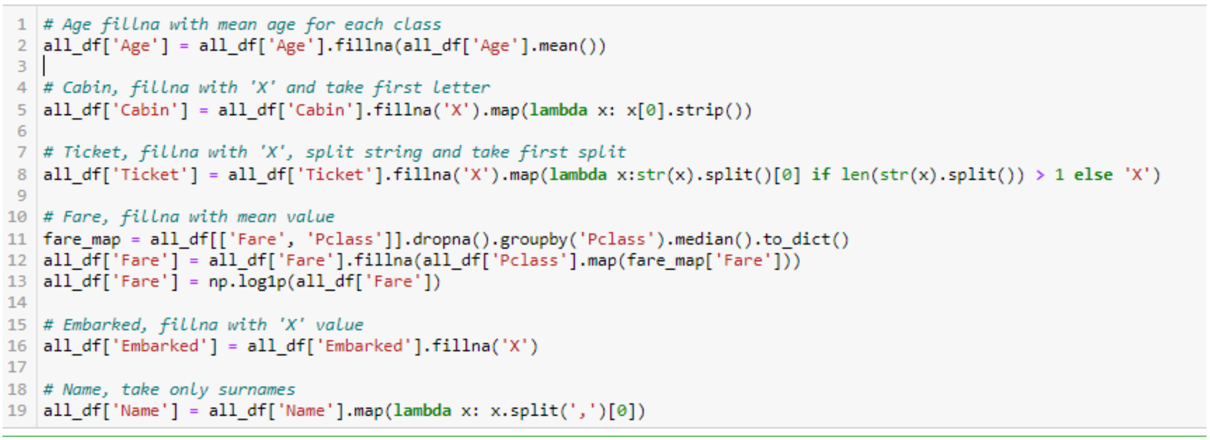
The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Step 2: Loading the datasets



```
train_df = pd.read_csv('../input/tabular-playground-series-apr-2021/train.csv')
test_df = pd.read_csv('../input/tabular-playground-series-apr-2021/test.csv')
submission = pd.read_csv('../input/tabular-playground-series-apr-2021/sample_submission.csv')
test_df[TARGET] = pd.read_csv("../input/tps-apr-2021-pseudo-labeling-voting-ensemble/voting_submission.csv")[TARGET]

all_df = pd.concat([train_df, test_df]).reset_index(drop=True)
```

Step 3: Filling the Missing values



```python
# Age fillna with mean age for each class
all_df['Age'] = all_df['Age'].fillna(all_df['Age'].mean())

# Cabin, fillna with 'X' and take first letter
all_df['Cabin'] = all_df['Cabin'].fillna('X').map(lambda x: x[0].strip())

# Ticket, fillna with 'X', split string and take first split
all_df['Ticket'] = all_df['Ticket'].fillna('X').map(lambda x:str(x).split()[0] if len(str(x).split()) > 1 else 'X')

# Fare, fillna with mean value
fare_map = all_df[['Fare', 'Pclass']].dropna().groupby('Pclass').median().to_dict()
all_df['Fare'] = all_df['Fare'].fillna(all_df['Pclass'].map(fare_map['Fare']))
all_df['Fare'] = np.log1p(all_df['Fare'])

# Embarked, fillna with 'X' value
all_df['Embarked'] = all_df['Embarked'].fillna('X')

# Name, take only surnames
all_df['Name'] = all_df['Name'].map(lambda x: x.split(',')[0])
```

Step 4: Encoding

Label Encoder : Encode target labels with value between 0 and n_classes -1.

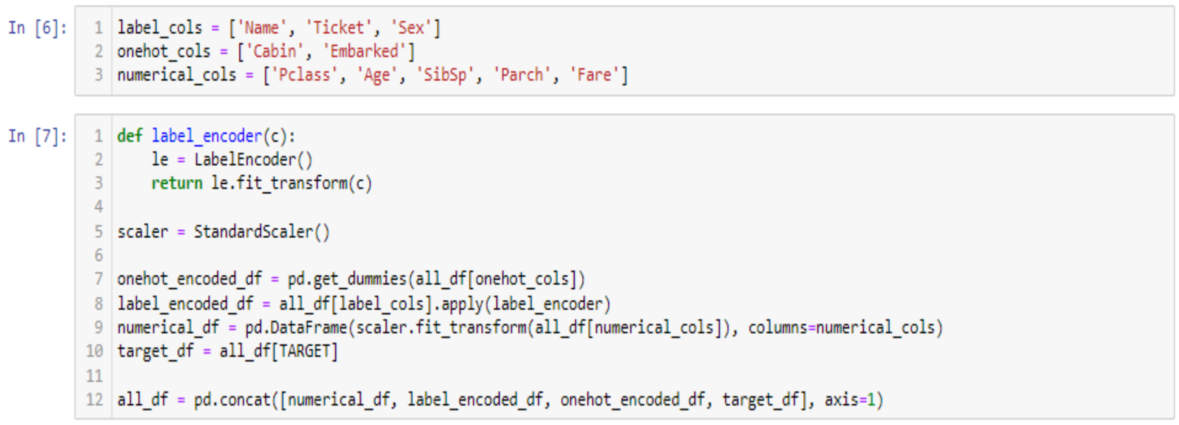
This transformer should be used to encode target values, *i.e.* y, and not the input X.

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as:

z = (x - u) / s

where u is the mean of the training samples or zero if with_mean=False, and s is the standard deviation of the training samples or one if with_std=False.

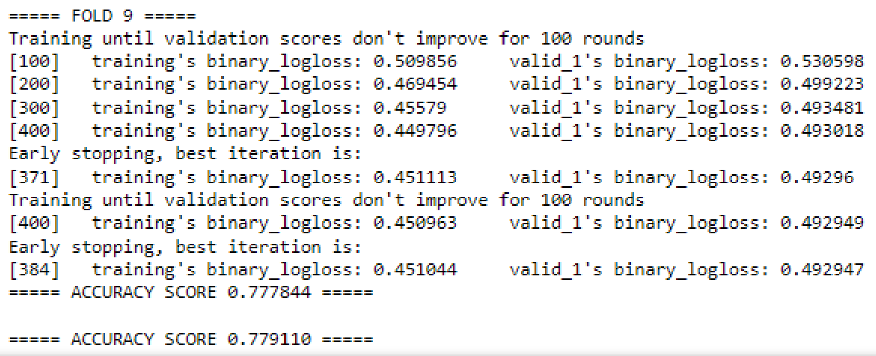


```python
In [6]:
label_cols = ['Name', 'Ticket', 'Sex']
onehot_cols = ['Cabin', 'Embarked']
numerical_cols = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

```python
In [7]:
def label_encoder(c):
    le = LabelEncoder()
    return le.fit_transform(c)

scaler = StandardScaler()

onehot_encoded_df = pd.get_dummies(all_df[onehot_cols])
label_encoded_df = all_df[label_cols].apply(label_encoder)
numerical_df = pd.DataFrame(scaler.fit_transform(all_df[numerical_cols]), columns=numerical_cols)
target_df = all_df[TARGET]

all_df = pd.concat([numerical_df, label_encoded_df, onehot_encoded_df, target_df], axis=1)
```

Step 5: LightGBM

- LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:
    - Faster training speed and higher efficiency.
    - Lower memory usage.
    - Better accuracy.
    - Support of parallel and GPU learning.
    - Capable of handling large-scale data.

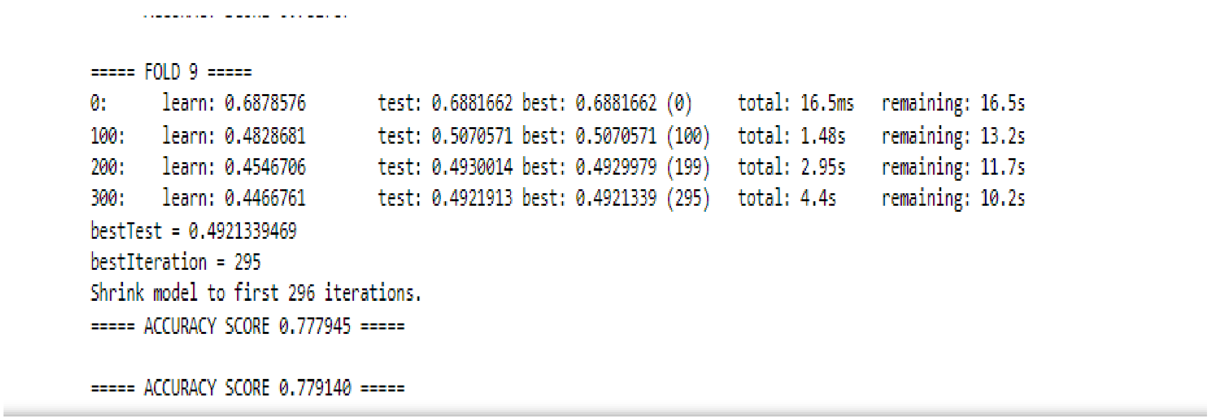


```
===== FOLD 9 =====
Training until validation scores don't improve for 100 rounds
[100]   training's binary_logloss: 0.509856     valid_1's binary_logloss: 0.530598
[200]   training's binary_logloss: 0.469454     valid_1's binary_logloss: 0.499223
[300]   training's binary_logloss: 0.45579      valid_1's binary_logloss: 0.493481
[400]   training's binary_logloss: 0.449796     valid_1's binary_logloss: 0.493018
Early stopping, best iteration is:
[371]   training's binary_logloss: 0.451113     valid_1's binary_logloss: 0.49296
Training until validation scores don't improve for 100 rounds
[400]   training's binary_logloss: 0.450963     valid_1's binary_logloss: 0.492949
Early stopping, best iteration is:
[384]   training's binary_logloss: 0.451044     valid_1's binary_logloss: 0.492947
===== ACCURACY SCORE 0.777844 =====

===== ACCURACY SCORE 0.779110 =====
```

Step 6: CatBoost

CatBoost or Categorical Boosting is an open-source boosting library developed by Yandex. In addition to regression and classification, CatBoost can be used in ranking, recommendation systems, forecasting and even personal assistants.

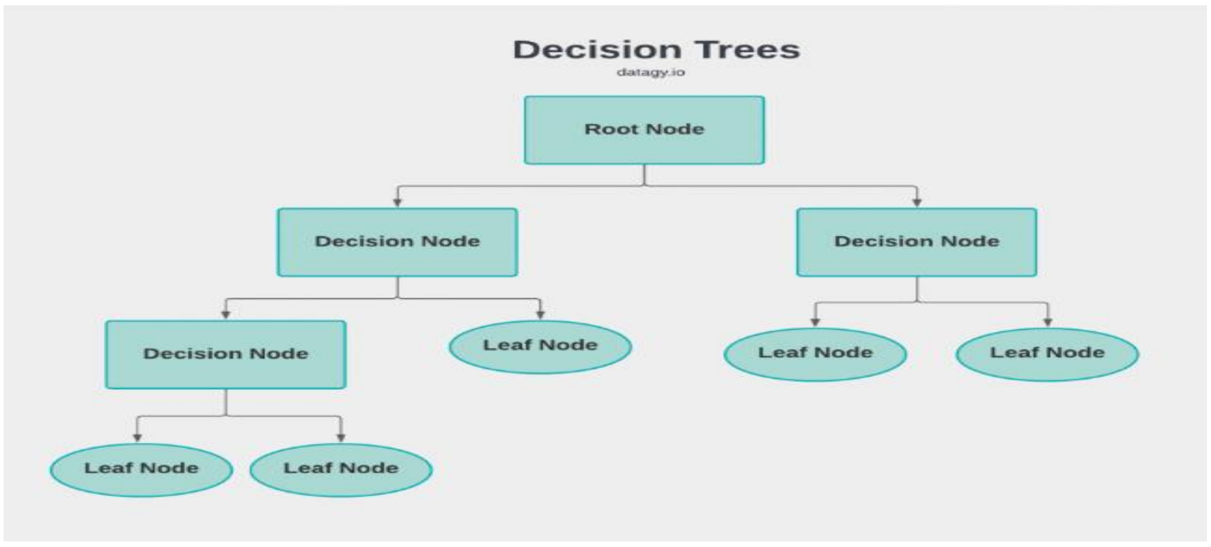
- It is especially powerful in two ways:
- 1.It yields state-of-the-art results without extensive data training typically required by other machine learning methods, and
- 2.Provides powerful out-of-the-box support for the more descriptive data formats that accompany many business problems.
- "CatBoost" name comes from two words
  - "Category" and "Boosting".

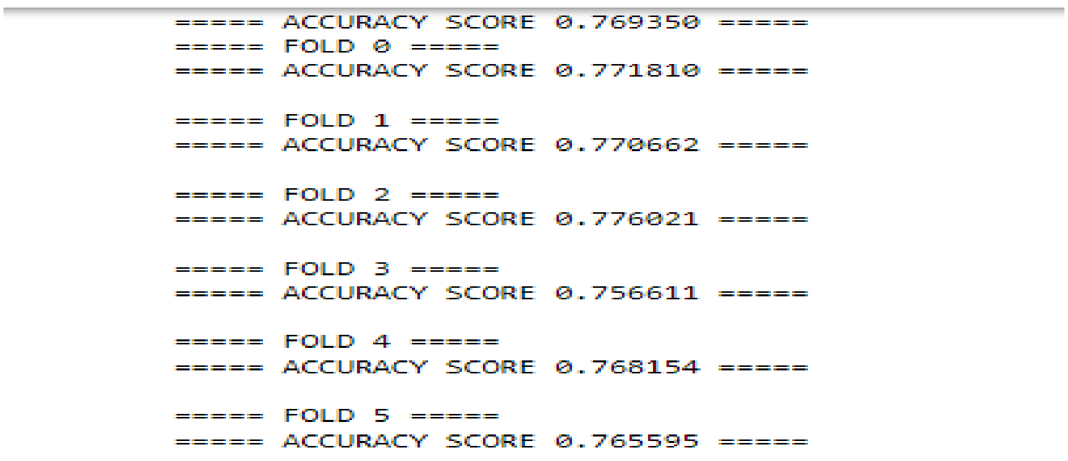


```
===== FOLD 9 =====
0:      learn: 0.6878576        test: 0.6881662 best: 0.6881662 (0)     total: 16.5ms   remaining: 16.5s
100:    learn: 0.4828681        test: 0.5070571 best: 0.5070571 (100)   total: 1.48s    remaining: 13.2s
200:    learn: 0.4546706        test: 0.4930014 best: 0.4929979 (199)   total: 2.95s    remaining: 11.7s
300:    learn: 0.4466761        test: 0.4921913 best: 0.4921339 (295)   total: 4.4s     remaining: 10.2s
bestTest = 0.4921339469
bestIteration = 295
Shrink model to first 296 iterations.
===== ACCURACY SCORE 0.777945 =====

===== ACCURACY SCORE 0.779140 =====
```

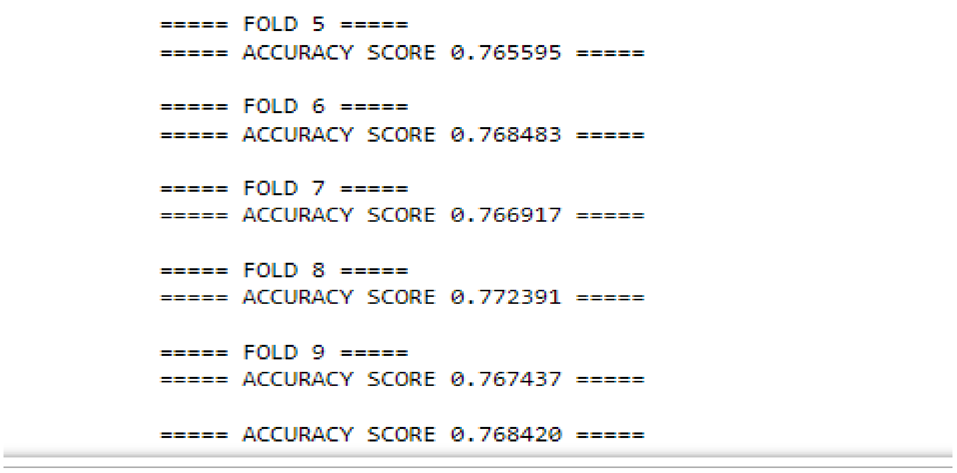
Step 7: Decision Tree Model

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.





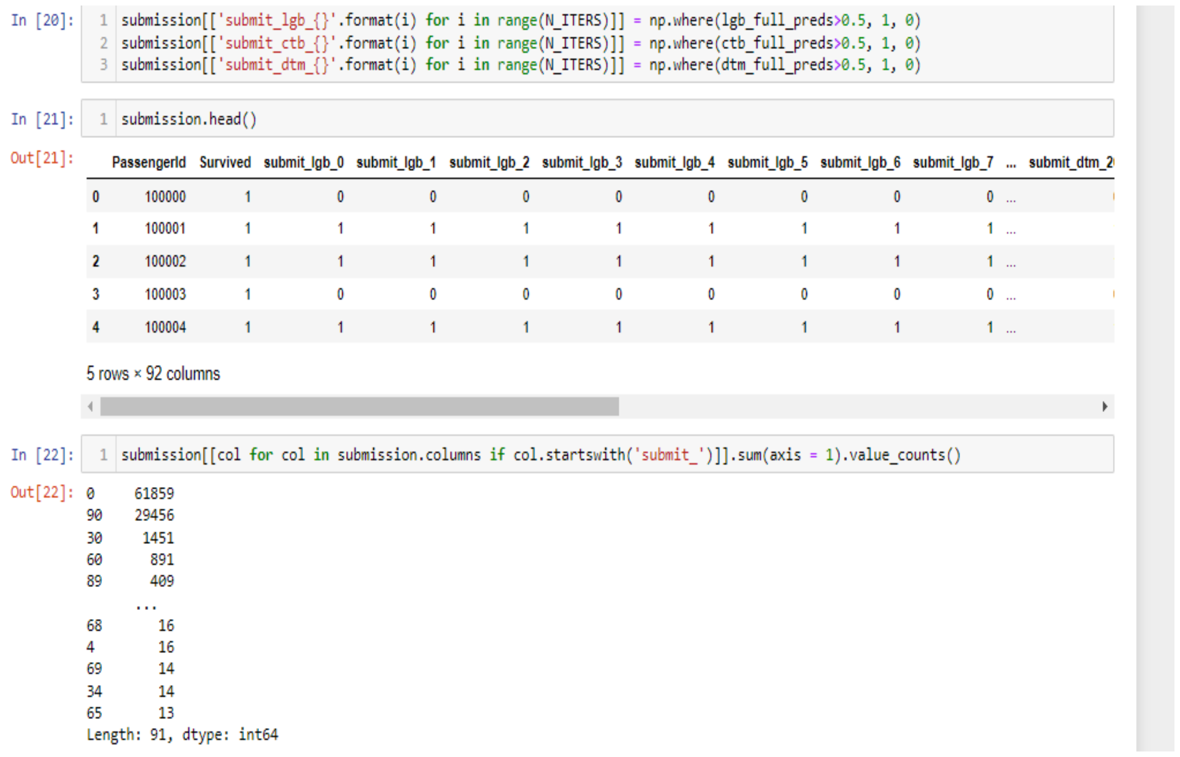
```
===== ACCURACY SCORE 0.769350 =====
===== FOLD 0 =====
===== ACCURACY SCORE 0.771810 =====

===== FOLD 1 =====
===== ACCURACY SCORE 0.770662 =====

===== FOLD 2 =====
===== ACCURACY SCORE 0.776021 =====

===== FOLD 3 =====
===== ACCURACY SCORE 0.756611 =====

===== FOLD 4 =====
===== ACCURACY SCORE 0.768154 =====

===== FOLD 5 =====
===== ACCURACY SCORE 0.765595 =====
```



```
===== FOLD 5 =====
===== ACCURACY SCORE 0.765595 =====

===== FOLD 6 =====
===== ACCURACY SCORE 0.768483 =====

===== FOLD 7 =====
===== ACCURACY SCORE 0.766917 =====

===== FOLD 8 =====
===== ACCURACY SCORE 0.772391 =====

===== FOLD 9 =====
===== ACCURACY SCORE 0.767437 =====

===== ACCURACY SCORE 0.768420 =====
```

Step 8: Ensemble

Ensemble means a group of elements viewed as a whole rather than individually. An Ensemble method creates multiple models and combines them to solve it. Ensemble methods help to improve the robustness/generalizability of the model.



```
In [20]: submission[['submit_lgb_{}'.format(i) for i in range(N_ITERS)]] = np.where(lgb_full_preds>0.5, 1, 0)
         submission[['submit_ctb_{}'.format(i) for i in range(N_ITERS)]] = np.where(ctb_full_preds>0.5, 1, 0)
         submission[['submit_dtm_{}'.format(i) for i in range(N_ITERS)]] = np.where(dtm_full_preds>0.5, 1, 0)

In [21]: submission.head()
```

Out[21]:

	PassengerId	Survived	submit_lgb_0	submit_lgb_1	submit_lgb_2	submit_lgb_3	submit_lgb_4	submit_lgb_5	submit_lgb_6	submit_lgb_7	...	submit_dtm_2
0	100000	1	0	0	0	0	0	0	0	0	...	
1	100001	1	1	1	1	1	1	1	1	1	...	
2	100002	1	1	1	1	1	1	1	1	1	...	
3	100003	1	0	0	0	0	0	0	0	0	...	
4	100004	1	1	1	1	1	1	1	1	1	...	

5 rows × 92 columns

```
In [22]: submission[[col for col in submission.columns if col.startswith('submit_')]].sum(axis = 1).value_counts()

Out[22]: 0     61859
         90    29456
         30     1451
         60      891
         89      409
                ...
         68       16
         4        16
         69       14
         34       14
         65       13
         Length: 91, dtype: int64
```